**📋 Estructura Optimizada de Archivos - Sistema SSO**

**🎯 ESTRUCTURA MINIMALISTA PERO COMPLETA**

**Enfoque:** Solo los archivos **ESENCIALES** para administrar todo desde la interfaz web, sin redundancias ni archivos innecesarios.

**📁 ESTRUCTURA DEL PROYECTO (OPTIMIZADA)**

microservicio-sso/

├── 📄 docker-compose.yml # Orquestación completa

├── 📄 package.json # Dependencias backend

├── 📄 .env.example # Variables de entorno

├── 📄 README.md # Documentación básica

│

├── 📁 src/ # Backend Node.js

│ ├── 📄 app.js # App principal + todas las rutas

│ ├── 📄 database.js # Conexión PostgreSQL + modelos

│ ├── 📄 auth.js # Todo lo de autenticación y JWT

│ ├── 📄 controllers.js # Todos los controladores en uno

│ └── 📄 utils.js # Utilidades y helpers

│

├── 📁 public/ # Frontend completo

│ ├── 📄 index.html # Login + Dashboard TODO EN UNO

│ ├── 📄 style.css # Todos los estilos en un archivo

│ └── 📄 app.js # Todo el JavaScript en uno

│

├── 📁 database/ # Base de datos

│ └── 📄 init.sql # Script completo de BD

│

├── 📁 docker/ # Docker config

│ ├── 📄 Dockerfile # Imagen única

│ ├── 📄 nginx.conf # Config Nginx

│ └── 📄 start.sh # Script inicio

│

└── 📁 sdk/ # SDK simple para microservicios

├── 📄 sso-client.js # Cliente Node.js completo

└── 📄 example-integration.js # Ejemplo de uso

**📋 ARCHIVOS DETALLADOS - FUNCIONALIDAD COMPLETA**

**🐳 ORQUESTACIÓN**

**📄 docker-compose.yml**

**Propósito:** Levantar todo el sistema con un comando **Contenido:**

* PostgreSQL con datos iniciales automáticos
* Redis para cache de sesiones
* Contenedor SSO principal (Node.js + Nginx + Frontend)
* Red Docker para comunicación
* Volúmenes persistentes para datos
* Health checks automáticos

**📄 docker/Dockerfile**

**Propósito:** Imagen única multi-stage **Contenido:**

* Base Node.js 18 Alpine
* Nginx para servir frontend
* Scripts de inicio automáticos
* Todo optimizado en una sola imagen

**📄 docker/nginx.conf**

**Propósito:** Proxy y servidor web **Contenido:**

* Servir / → Frontend estático
* Proxy /api/\* → Backend Node.js
* Headers de seguridad
* Configuración optimizada

**📄 docker/start.sh**

**Propósito:** Inicio automático del sistema **Contenido:**

* Esperar PostgreSQL
* Ejecutar migraciones
* Crear admin inicial
* Iniciar Nginx + Node.js

**🔧 BACKEND - TODO EN 4 ARCHIVOS**

**📄 src/app.js**

**Propósito:** Aplicación Express completa con TODAS las rutas **Contenido:**

// Configuración Express + middlewares

// TODAS las rutas de la API:

// ✅ POST /api/auth/login, /logout, /verify

// ✅ GET/POST/PUT/DELETE /api/users (CRUD completo)

// ✅ GET/POST/PUT/DELETE /api/roles (CRUD completo)

// ✅ GET/POST/PUT/DELETE /api/permissions (CRUD completo)

// ✅ GET/POST/PUT/DELETE /api/services (CRUD completo)

// ✅ GET /api/dashboard/stats (métricas dashboard)

// ✅ GET /api/monitoring/logs (logs en tiempo real)

// ✅ GET /api/health (health check)

// ✅ POST /api/services/register (auto-registro)

// ✅ GET /api/services/:id/routes (endpoints por servicio)

**📄 src/database.js**

**Propósito:** Conexión PostgreSQL + todos los modelos **Contenido:**

// Conexión a PostgreSQL

// Funciones para TODOS los modelos:

// ✅ Users: create, getById, getAll, update, delete, assignRole

// ✅ Roles: create, getById, getAll, update, delete, assignPermissions

// ✅ Permissions: create, getById, getAll, update, delete

// ✅ Services: register, getAll, updateStatus, getRoutes

// ✅ UserSessions: create, validate, revoke

// ✅ AuditLogs: create, getByUser, getAll

// ✅ Dashboard: getStats, getRecentActivity, getAlerts

**📄 src/auth.js**

**Propósito:** Todo lo relacionado con autenticación y seguridad **Contenido:**

// JWT: generate, verify, refresh tokens

// Password: hash, compare con bcrypt

// Middleware: requireAuth, requirePermission

// Session management: create, validate, revoke

// Permission checking: hasPermission, getUserPermissions

// Rate limiting y security headers

**📄 src/controllers.js**

**Propósito:** Lógica de negocio para todas las APIs **Contenido:**

// authController: login, logout, verify, refresh

// userController: createUser, getUsers, updateUser, deleteUser, assignRoles

// roleController: createRole, getRoles, updateRole, deleteRole, assignPermissions

// serviceController: registerService, getServices, updateService, getRoutes

// dashboardController: getStats, getActivity, getAlerts

// monitoringController: getLogs, getMetrics, getServiceHealth

**📄 src/utils.js**

**Propósito:** Utilidades y helpers del sistema **Contenido:**

// Response formatters: success(), error(), paginate()

// Validators: validateEmail, validatePassword, validatePermission

// Logger: log info, warn, error

// Crypto helpers: generateToken, hashPassword

// Constants: default roles, permissions, settings

**🎨 FRONTEND - TODO EN 3 ARCHIVOS**

**📄 public/index.html**

**Propósito:** TODA la interfaz web en un solo archivo HTML **Contenido:**

<!-- SECCIÓN LOGIN (inicial) -->

<div id="loginSection">

<!-- Formulario de login elegante -->

</div>

<!-- SECCIÓN DASHBOARD PRINCIPAL -->

<div id="dashboardSection" style="display:none">

<!-- Sidebar de navegación -->

<!-- Métricas en tiempo real -->

<!-- Centro de alertas -->

</div>

<!-- SECCIÓN GESTIÓN DE USUARIOS -->

<div id="usersSection" style="display:none">

<!-- Tabla de usuarios con búsqueda -->

<!-- Modal crear/editar usuario -->

<!-- Asignación de roles drag & drop -->

</div>

<!-- SECCIÓN GESTIÓN DE ROLES -->

<div id="rolesSection" style="display:none">

<!-- Lista de roles -->

<!-- Editor visual de permisos (matriz checkbox) -->

<!-- Modal crear/editar rol -->

</div>

<!-- SECCIÓN GESTIÓN DE SERVICIOS -->

<div id="servicesSection" style="display:none">

<!-- Lista de microservicios conectados -->

<!-- Estado en tiempo real -->

<!-- Configuración de endpoints -->

<!-- Auto-descubrimiento de rutas -->

</div>

<!-- SECCIÓN MONITOREO -->

<div id="monitoringSection" style="display:none">

<!-- Métricas en tiempo real -->

<!-- Logs del sistema -->

<!-- Gráficos de actividad -->

</div>

<!-- SECCIÓN CONFIGURACIÓN -->

<div id="settingsSection" style="display:none">

<!-- Configuración del sistema -->

<!-- Backup y restauración -->

<!-- Configuración de email -->

</div>

<!-- MODALES REUTILIZABLES -->

<div id="modalContainer"></div>

**📄 public/style.css**

**Propósito:** TODOS los estilos del sistema en un archivo **Contenido:**

/\* Reset y variables CSS \*/

/\* Estilos del login \*/

/\* Estilos del dashboard y sidebar \*/

/\* Estilos de tablas y listas \*/

/\* Estilos de formularios y modales \*/

/\* Estilos de botones y componentes \*/

/\* Estilos responsive \*/

/\* Animaciones y transiciones \*/

/\* Tema dark/light \*/

**📄 public/app.js**

**Propósito:** TODA la lógica JavaScript del frontend **Contenido:**

// 🔐 AUTH MANAGER

class AuthManager {

login(), logout(), checkAuth(), refreshToken()

}

// 🌐 API CLIENT

class ApiClient {

request(), get(), post(), put(), delete()

// Métodos específicos para cada endpoint

}

// 🎛️ DASHBOARD MANAGER

class DashboardManager {

loadStats(), loadAlerts(), loadActivity(), updateMetrics()

}

// 👥 USER MANAGER

class UserManager {

loadUsers(), createUser(), editUser(), deleteUser(), assignRoles()

showUserModal(), handleDragDrop()

}

// 🎭 ROLE MANAGER

class RoleManager {

loadRoles(), createRole(), editRole(), deleteRole()

showPermissionMatrix(), updatePermissions(), cloneRole()

}

// 🔧 SERVICE MANAGER

class ServiceManager {

loadServices(), registerService(), updateService()

autoDiscoverRoutes(), configureEndpoints(), monitorHealth()

}

// 📊 MONITORING MANAGER

class MonitoringManager {

loadLogs(), loadMetrics(), startRealTimeUpdates()

showGraphs(), filterLogs()

}

// 🛠️ UI MANAGER

class UIManager {

showSection(), hideSection(), showModal(), closeModal()

showNotification(), updateBreadcrumb(), handleNavigation()

}

// 🚀 APP INITIALIZATION

document.addEventListener('DOMContentLoaded', () => {

// Inicializar todas las clases

// Configurar event listeners

// Verificar autenticación inicial

// Mostrar sección apropiada

});

**🗄️ BASE DE DATOS**

**📄 database/init.sql**

**Propósito:** Script COMPLETO de inicialización **Contenido:**

-- Crear TODAS las tablas con relaciones

-- Insertar roles del sistema (Super Admin, Admin, Manager, User)

-- Insertar TODOS los permisos del SSO

-- Crear usuario admin por defecto (admin@sso.com / admin123)

-- Asignar permisos a roles

-- Crear índices optimizados

-- Crear triggers de auditoría

-- Insertar datos de ejemplo

-- Funciones SQL para validaciones rápidas

**📦 SDK PARA MICROSERVICIOS**

**📄 sdk/sso-client.js**

**Propósito:** Cliente completo para integrar microservicios **Contenido:**

class SSOClient {

constructor(options)

// Middleware Express

middleware()

requirePermission(permission)

// Auto-registro

register()

discoverEndpoints()

// Validación

verifyToken(token)

getUserPermissions(userId)

// Health check

sendHeartbeat()

}

module.exports = SSOClient;

**📄 sdk/example-integration.js**

**Propósito:** Ejemplo práctico de integración **Contenido:**

// Ejemplo completo de cómo integrar un microservicio

const express = require('express');

const SSOClient = require('./sso-client');

const app = express();

const sso = new SSOClient({

ssoUrl: 'http://sso:3000',

serviceName: 'my-service'

});

// Aplicar middleware de autenticación

app.use('/api', sso.middleware());

// Rutas con permisos específicos

app.get('/api/data', sso.requirePermission('data.read'), (req, res) => {

res.json({ user: req.user, data: 'protected data' });

});

// Auto-registrar servicio

sso.register();

**🎯 FUNCIONALIDADES INCLUIDAS EN ESTOS ARCHIVOS**

**✅ Gestión Completa de Usuarios**

* Crear, editar, eliminar usuarios desde la interfaz
* Asignación visual de roles (drag & drop)
* Ver actividad de usuarios en tiempo real
* Suspender/activar usuarios con un clic

**✅ Gestión Visual de Roles**

* Crear roles personalizados
* Editor visual de permisos (matriz de checkboxes)
* Clonar roles existentes
* Ver estadísticas de uso de roles

**✅ Gestión de Microservicios**

* Auto-registro de servicios cuando arrancan
* Configuración visual de endpoints y permisos
* Auto-descubrimiento de rutas
* Monitoreo de salud en tiempo real

**✅ Dashboard en Tiempo Real**

* Métricas actualizadas automáticamente
* Centro de alertas
* Gráficos de actividad
* Estado de todos los servicios

**✅ Monitoreo y Auditoría**

* Logs del sistema en tiempo real
* Historial de acciones de usuarios
* Métricas de performance
* Alertas automáticas

**✅ Configuración del Sistema**

* Configurar políticas de seguridad
* Gestionar notificaciones por email
* Backup y restauración
* Configuración de tokens JWT

**🚀 PROCESO DE DESARROLLO**

**Orden de Implementación:**

1. **📄 docker-compose.yml** - Configurar orquestación
2. **📄 database/init.sql** - Crear esquema y datos
3. **📄 src/database.js** - Conexión y modelos
4. **📄 src/auth.js** - Sistema de autenticación
5. **📄 src/controllers.js** - Lógica de negocio
6. **📄 src/app.js** - APIs y rutas
7. **📄 public/index.html** - Interfaz completa
8. **📄 public/style.css** - Estilos visuales
9. **📄 public/app.js** - Lógica frontend
10. **📄 sdk/sso-client.js** - SDK para microservicios

**Resultado Final:**

* **Un solo comando:** docker-compose up -d
* **Acceso inmediato:** http://localhost:3000
* **Login admin:** admin@sso.com / admin123
* **Control total:** Desde la interfaz web
* **Auto-registro:** Microservicios se conectan solos

**Beneficios de esta Estructura:**

* ✅ **Desarrollo rápido:** Solo 12 archivos principales
* ✅ **Fácil mantenimiento:** Todo centralizado
* ✅ **Funcionalidad completa:** Sin sacrificar características
* ✅ **Deploy simple:** Un solo contenedor
* ✅ **Escalable:** Fácil agregar funciones nuevas